Task-Node Mapping in an Arbitrary Computer Network using SMT Solver*

Andrii Kovalov, Elisabeth Lobe, Andreas Gerndt, and Daniel Lüdtke

German Aerospace Center (DLR), Simulation and Software Technology
{andrii.kovalov,elisabeth.lobe,andreas.gerndt,daniel.luedtke}@dlr.de

Abstract. The problem of mapping (assigning) application tasks to processing nodes in a distributed computer system for spacecraft is investigated in this paper. The network architecture is developed in the project 'Scalable On-Board Computing for Space Avionics' (ScOSA) at the German Aerospace Center (DLR). In ScOSA system the processing nodes are connected to a network with an arbitrary topology. The applications are structured as directed graphs of periodic and aperiodic tasks that exchange messages. In this paper a formal definition of the mapping problem is given. We demonstrate several ways to formulate it as a satisfiability modulo theories (SMT) problem and then use Z3, a state-of-the-art SMT solver, to produce the mapping. The approach is evaluated on a mapping problem for an optical navigation application as well as on a set of randomly generated task graphs.

1 Introduction

In this paper we investigate the generation of optimal task-node mappings for a distributed system structured as a collection of communicating tasks which run on a computer network with an arbitrary topology. Our work is motivated by the existing project 'Scalable On-Board Computing for Space Avionics' (ScOSA) at the German Aerospace Center (DLR), however the proposed approach is not specific to this project and can be used for task-node mapping in various contexts.

1.1 ScOSA System Overview

The conventional way to design on-board computers for spacecraft is to have independent dedicated computers for various subsystems (payload, communication, attitude control, etc.) including the redundant counterparts for fault-tolerance. This conventional approach does not scale well under the increasing demands on the on-board data processing capabilities.

The project ScOSA at DLR aims at developing a scalable on-board computer system [15]. Most importantly, in a ScOSA system all computers are connected to a network that provides the computing power to all the on-board applications.

* The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-66845-1_12

The applications for ScOSA are organized as directed graphs of tasks that send messages. An example of a task graph and a network structure is shown in Fig. 1.

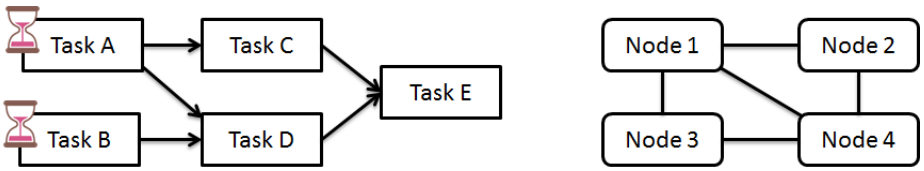

Fig. 1. Example task graph (left) and network (right)

In this example the first tasks (A and B) run periodically with certain periods. When they are finished, they send messages to the next tasks, which are triggered by these incoming messages, and so on. The tasks start either when they receive all messages that are not *optional*, or when they receive a *final* message, which triggers a task to start immediately. Every task is assigned to a particular processing node in the network. The messages can either be passed locally or sent over the network if the destination task is located on another node.

One of the important concepts in ScOSA is reconfiguration. There are special nodes ('Master' and 'Observers') that constantly monitor the state of the other nodes. If a failure is detected, a reconfiguration command is broadcast to the network, and the tasks are reassigned to the healthy nodes.

It was decided that ScOSA does not employ adaptive reconfiguration at runtime [15]. Instead, in order to achieve a higher degree of deterministic behavior, all possible configurations with their respective activation events have to be determined before launch of the spacecraft or during updates uploaded by ground control. In case of a failure, ground control needs to be able to understand and reproduce the behavior of the system, including the conducted reconfigurations, in order to resume normal operation of the spacecraft. As there could be many combinations of consecutive node failures, there is a need to automatically generate a large number of task-node mappings.

1.2 Mapping and Subsequent Model Checking

Our goal is to have a two-stage process shown in Fig. 2 where we first generate a mapping and then verify that this mapping cannot lead the system into an undesirable state. In this paper we only discuss the mapping problem; model checking will be the next step in our work.

Our approach is to formulate the mapping problem as a satisfiability modulo theories (SMT) problem, and then use a state-of-the-art optimizing SMT solver to solve it. We chose this approach instead of using heuristics (such as greedy algorithms) because of two reasons. First of all, it guarantees the optimal solution,

and for our purposes we would prefer the best solution to a fast but suboptimal one. Secondly, we can later introduce some bounded model checking into the SMT-based mapping, for example to check that the first few runs of the task graph do not block the network. In the subsequent model checking step, we are planning to analyze more complex properties on a more detailed system model.

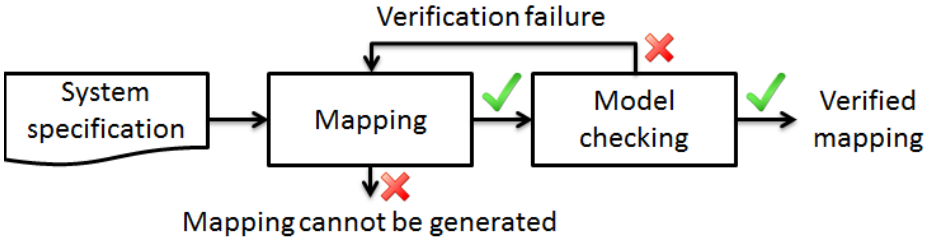

Fig. 2. Our proposed approach to generate and verify the mapping (in this paper only the mapping generation is discussed, model checking is future work).

The rest of the paper is structured as follows. Sect. 2 gives an overview of the existing work in task-node mapping, especially with the use of SMT solvers. In Sect. 3 we give a formal definition of our task-node mapping problem, which we then formulate as an SMT problem in Sect. 4. The approach is evaluated in Sect. 5, and Sect. 6 concludes the paper.

2 Related Work

The problem of mapping application tasks to hardware processing nodes is an important practical problem and has been studied for decades. Several formulations of it have been shown to be NP-complete [1,10], so it is usually solved by applying heuristics. A good overview of mapping algorithms is given in [9]. One of the widely used heuristics is a greedy heuristic. Lee and Aggarwal [11] show an approach where the initial mapping is produced with a greedy algorithm, and then improved with a series of pairwise exchanges. Other greedy algorithms are shown in [5] and [8]. The latter presents a hierarchical algorithm which treats the hardware as a tree and for each level of this tree divides the tasks into groups according to the communication patterns among them. Another algorithm [6] partitions the hardware graph based on the coordinates of the hardware nodes, which is practical for multiprocessor systems that have a grid or torus topology.

Genetic algorithms can also be used to search for an optimal mapping. In [12] a two-step genetic algorithm is shown that first maps the tasks to a set of node types using an average communication time, and then maps the tasks to the specific nodes considering the actual communication time between them. Another genetic algorithm [20] solves a slightly different problem, where not all

the tasks are mapped to the nodes, but the ones that give the most value to the system.

The task-node mapping problem can also be addressed with the exhaustive search tools such as integer linear programming (ILP) or satisfiability modulo theories (SMT) solvers. In [1] and [17] an ILP-based approach is shown where the task-node mapping problem is first formulated as an integer linear program and then relaxed to a linear program (LP) which can be solved in polynomial time. The solution to the LP problem maps a subset of tasks to the nodes, and the exhaustive enumeration is used to map the remaining tasks (which is exponential in the number of nodes). Another work [21] uses an ILP approach to solve the problem of mapping and scheduling together, focusing on the network contention (competition of different messages for a physical link).

There are some SMT-based approaches to the task mapping problem [18,19]. In [19] the use of symmetries is explored to reduce search space. In [18] the problem is divided into the master problem of producing a mapping, and sub-problems for scheduling. In order to explore the whole search space and find the optimal solution, the previously found solutions are learned and excluded. An improvement of this approach in [13] and [14] integrates the analysis of the current partial mapping into the SAT solving process itself.

A somewhat similar approach is described in [2] where first, the mapping problem is solved with answer set programming, then the solution is refined for better schedulability (e.g. load balancing), and then the schedule is produced with an SMT solver.

A slightly different application model is taken in [4] where tasks are not strictly mapped to nodes, but can be executed on different nodes according to an SMT-generated schedule.

3 Formal Definition of the Problem

This section gives the formal definition of the task-node mapping problem.

3.1 Period Estimation for Event-Triggered Tasks

For our formal definition we treat all the tasks as periodic. However, a ScOSA application can consist of both periodic and aperiodic tasks, therefore we first need to estimate the periods of the event-triggered tasks. The periods of the time-triggered tasks are known. Then, if a task runs with period p, it sends an output message with period p. We assume the receiving task gets this message with period p and therefore runs with the same period.

If a task waits for multiple messages, it only can start when all the necessary messages have arrived. In this case its period is defined by the longest period of all the incoming messages. With this simple approach we can propagate the period information to all the tasks as illustrated in Fig. 3. This period propagation only works for task graphs that are either acyclic or all the backward messages are optional and do not affect the task periods. In real applications the task graphs

can contain cycles, and the period estimation becomes more complex, especially with the use of *final* messages. Then, the task periods can be measured directly on the actual running system or in a simulation, but this is out of scope of this paper.

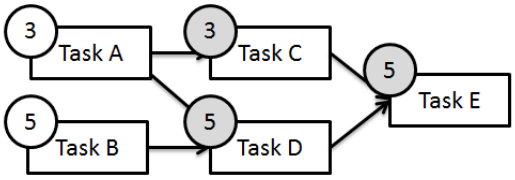

Fig. 3. Estimated periods of all the tasks in a task graph

3.2 Formulation of the Mapping Problem

Our formal definition of the problem is close to the one given in [10], but we also consider task utilization. The task graph is a directed graph (not necessarily connected) $TG = (T, E^T)$, where T is a set of tasks, and $E^T \subseteq T \times T$ is a set of edges representing message passing between the tasks.

Every task t has its worst case execution time $wcet_t$, its period p_t and the size of its output message m_t:

$$\forall t \in T : wcet_t > 0, p_t > 0, m_t \geq 0.$$

The utilization of the task t is then defined as $u(t) = wcet_t/p_t$. Utilization of the task is the fraction of its period when it is executing (the worst case time is chosen because we are targeting real-time systems).

Every task t that has outgoing edges in the task graph produces messages of size m_t greater than zero. We can say that all the other tasks (without outgoing edges) produce messages of zero size:

$$\forall t \in T : \begin{cases} m_t > 0, & \exists t' \in T : (t, t') \in E^T \\ m_t = 0, & \text{otherwise.} \end{cases}$$

The traffic tt between two tasks t and t' is then defined as

$$tt(t, t') = \begin{cases} 0, & (t, t') \notin E^T \\ m_t/p_t, & \text{otherwise} \end{cases}$$

and shows how much data task t is sending to task t' per time unit.

The communication network is represented as a graph $HN = (N, E)$ where N is a set of nodes and $E \subseteq \{\{n_1, n_2\} : n_1, n_2 \in N, n_1 \neq n_2\}$ is a set of network links. Every edge $e \in E$ has a bandwidth $b_e > 0$ which defines how much data

this link can transfer in one time unit. In reality the actual bandwidth of a link depends on the number and size of the messages that are transferred via that link. For few large messages, a link usually shows higher bandwidth than for many small messages due to the overhead from packet headers.

Not all tasks can be executed on all nodes. For every task there is a subset of nodes that can execute it. We define this with a function of available nodes $an : T \to 2^N$. For a task t, $an(t) \subseteq N$ is a set of nodes that can execute t.

Next we consider routing. Every router in the network has a routing table that specifies the output ports for all possible destination nodes. For example, the network in Fig. 1 has no link between Node 2 and Node 3. One way to route messages from 2 to 3 would be via Node 4. In this case the routing table of the router in Node 2 would contain an entry saying that all messages for Node 3 should be sent to Node 4. We assume that the routing is consistent and is given as input. Formally, routing is defined as $R \in \{0,1\}^{(N\times N)\times E}$, with

$$R_{n,n',e} = \begin{cases} 1, & \text{if the data sent from node } n \text{ to node } n' \text{ passes edge } e \\ 0, & \text{otherwise.} \end{cases}$$

If a message is sent from node n to node n' that have a direct link between them, then $R_{n,n',e} = 1$ if and only if $e = \{n, n'\}$.

Our aim is to find a mapping function $m : T \to N$, where for a task t, $m(t) \in an(t)$ is a node on which the task is running.

We consider two additional constraints on this mapping function: on the network bandwidth and on the node load. Mappings should be avoided where the amount of traffic transferred on a link exceeds its bandwidth. Mappings where too many tasks are mapped to a single node should also be excluded.

For the bandwidth constraint we first define traffic between two nodes n and n' as

$$nn(n, n') = \sum_{\substack{t,t'\in T \\ m(t)=n \\ m(t')=n'}} tt(t, t').$$

The amount of traffic that passes through an edge e is

$$\mathit{traffic}(e) = \sum_{n,n'\in N} R_{n,n',e} \cdot nn(n, n').$$

The constraint on the edge bandwidth is then written as

$$\forall e \in E : \mathit{traffic}(e) \leq b_e.$$

The load on a node n is defined as

$$\mathit{load}(n) = \sum_{\substack{t\in T \\ m(t)=n}} u(t).$$

The constraint on the node load is then formulated as

$$\forall n \in N : \mathit{load}(n) \leq L$$

where L is the node load limit, which is some constant less than 1. This constant should allow for context switching and other overhead on the node that is not directly related to task execution. The constraint on node load might be more restrictive depending on the actual scheduling policy on the node (particularly, on preemption), but in our model we assume that if the mapping satisfies this constraint, the tasks are schedulable on the node.

In order to have a quality measure for a mapping, we use two objectives. The first objective is to minimize the total amount of traffic in the network *totalTraffic* with

$$totalTraffic = \sum_{e \in E} traffic(e).$$

The second objective is to minimize the maximal load among all the nodes *maxLoad* with

$$maxLoad = \max_{n \in N}(load(n)).$$

This objective aims to distribute the load evenly across all nodes.

There are of course other possible objectives for the mapping generation. For example, the fastest execution of the whole task graph. This would make the system respond to the external changes quicker. We could also have application-specific objectives, such as dedicate more resources for critical tasks.

4 SMT Formulation

To produce an optimal mapping with respect to the chosen objective, we use a state-of-the-art SMT solver Z3 [16], which has built-in optimization functionality [3]. This means the solver produces not just any satisfying assignment but an assignment that maximizes or minimizes a given objective function.

In this section we show how we formulated the problem of finding a task-node mapping as an SMT problem. In some cases we developed alternative ways of modeling particular aspects of the problem. We compare the performance of alternative approaches in Sect. 5.

4.1 Task-Node Mapping Variables

First of all, we need to create variables that define the mapping of tasks to nodes, which will be assigned as a solution to the mapping problem. We tried two alternative approaches - using boolean variables and integer variables.

Boolean Mapping Variables. With this approach we create $|T| \times |N|$ boolean variables TN_{tn}, where if this variable is *true* then task t is mapped to node n and *false* otherwise.

In order to make sure that a task is mapped to exactly one node, we include the following assertion for every $t \in T$:

$$\bigvee_{n \in N} \left(TN_{tn} \wedge \bigwedge_{n' \in N \setminus \{n\}} \neg TN_{tn'} \right).$$

Additionally, we need to constrain the available nodes for every task t with the following assertion:

$$\bigvee_{n \in an(t)} TN_{tn}.$$

Integer Mapping Variables. Another approach is to create one integer variable TN_t for every task t that is equal to index i of the mapped node $m(t)$. For every TN_t the constraints are defined with following assertions:

$$(TN_t \geq 1) \wedge (TN_t \leq |N|),$$

$$\bigvee_{n_i \in an(t)} (TN_t = i).$$

In the rest of the section, we will use boolean mapping variables. The corresponding expressions for integer mapping variables are easy to formulate.

4.2 Link Bandwidth Constraints

To formulate the link bandwidth constraints, we first convert the task-task traffic into node-node traffic. The amount of traffic between the tasks is known and given as input, but the actual traffic at the links depends on the mapping.

$|N| \times |N|$ variables $NNtraffic_{nn'}$ represent the amount of traffic from node n to node n':

$$NNtraffic_{nn'} = \begin{cases} 0, & n = n' \\ \sum_{t \in T} \sum_{t' \in T} \text{ite}(TN_{tn} \wedge TN_{t'n'}, tt(t,t'), 0), & n \neq n', \end{cases}$$

where TN_{tn} is a boolean mapping variable, $tt(t,t')$ is the amount of traffic from task t to task t', and ite() is an 'if-then-else' function, which returns the second or the third argument depending on whether the first argument is true or false.

The link traffic for every link $e \in E$ is represented with variable $Ltraffic_e$, which is the sum of all node-node traffic amounts, where the route between the nodes passes e. This amount should not exceed the bandwidth of the link:

$$Ltraffic_e = \sum_{\substack{(n,n') \in N \times N \\ R_{n,n',e}=1}} NNtraffic_{nn'},$$

$$Ltraffic_e \leq b_e.$$

As mentioned, the real bandwidth decreases with the increase of the number of messages that are passed via the link. For applications with many small messages, this can be modeled by counting the number of messages contributing to the link traffic, and then decreasing the bound depending on this number.

4.3 Node Load Constraints

We represent the load on the nodes with $|N|$ variables $NodeLoad_n$ that are defined for every node n as follows:

$$NodeLoad_n = \sum_{t \in T} ite(TN_{tn}, u(t), 0),$$

$$NodeLoad_n \leq L,$$

where L is a constant defining the limit on the node load. It is also possible to assign individual load limits to different nodes to model nodes with different performance.

4.4 Objective Functions

The two objectives that we consider are the minimization of the total network traffic and the minimization of the highest node load (load balancing). The first objective function can be formulated as follows

$$\sum_{e \in E} Ltraffic_e.$$

For the second objective we need to minimize the highest node load among all the nodes. We first define the function $\max(x, y)$, and then use it to get the highest load among all nodes

$$\max(x, y) = ite(x > y, x, y)$$

The highest node load is defined as follows and can then be minimized

$$maxNodeLoad$$
$$= \max(NodeLoad_1, \max(NodeLoad_2, \ldots \max(NodeLoad_{m-1}, NodeLoad_m) \ldots))$$

4.5 Variable Types

Another practical question of SMT formulation is what type of variables to use for numerical values, as Z3 supports both real and integer variables. In our model we have numeric values for traffic amounts and task utilization values. Our first approach is to encode everything with real variables with their actual values.

Another approach would be to round all the values and encode them as integers. However, for the task utilization it is not practical because the values are between 0 and 1. Instead of just rounding, we multiply the real values by a factor such as 100 or 1000, and then round them. Depending on the factor, we might lose precision and, presumably, the performance might be different. We evaluate both approaches, and discuss the results in the following section.

5 Evaluation

In this section we compare the task-node mapping performance with different variations of the SMT formulation, show the application of our mapping approach to an existing optical navigation system, and estimate the scalability of the approach. All the described experiments were performed with an off-the-shelf laptop with 16 GB RAM and a quad-core 2.6 GHz processor using Z3 version 4.5.0 on generated SMT-LIB files (without explicitly specified logic).

5.1 Performance Comparison of Different SMT Formulations

In order to understand how the runtime of Z3 depends on the SMT formulation, we took several random task graphs and solved them with all the SMT formulations discussed in the previous section for both our objective functions using the four-node network from Fig. 1. We took three task graphs with 10 tasks and three task graphs with 15 tasks, all of which were satisfiable. In each of these two groups there was one instance with a low load on the network (total task utilization of around 25% of the maximum load; total traffic in the task graph around 25% of the total bandwidth summed over all links), one with medium load (task utilization and total traffic of around 50%), and one with high load (task utilization and total traffic of around 75%). The running times are shown in Table 1. The rows show different SMT formulations, and the columns show different mapping problem instances. The SMT formulations differ in the type of mapping variables (see Subsect. 4.1) and the type of numerical variables (see Sect. 4.5).

Table 1. Mapping time in seconds of different SMT encodings for instances with different numbers of tasks (10 tasks and 15 tasks) and different loads on the network (L - low load, M - medium load, H - high load, around 25, 50, 75 percent of maximum load respectively). # - incorrect mapping, * - suboptimal mapping.

Mapping variables	Numeric variables	Traffic Minimization						Node load minimization					
		10 tasks			15 tasks			10 tasks			15 tasks		
		L	M	H	L	M	H	L	M	H	L	M	H
Boolean	Int(*100)	0.1	2.6	0.7	(#)0.2	(#)0.5	0.9	(*) 6.8	2.8	2.9	2.9	10.2	(*) 17.1
	Int(*1000)	0.2	1.4	0.7	0.3	3.5	1.2	10.7	4.8	3.4	17.8	25.3	91.5
	Int(*10000)	0.2	0.3	1.0	0.5	23.6	1.4	10.9	4.5	4.4	38.8	67.9	109.9
	Real	1.0	0.3	1.6	0.5	29.6	1.9	12.8	7.3	5.3	39.5	92.0	200.5
Integer	Int(*100)	0.2	1.1	0.8	(#)1.2	(#)2.1	1.1	(*) 7.2	3.5	3.7	3.6	11.3	(*) 25.4
	Int(*1000)	0.9	1.4	1.0	1.9	11.2	1.6	12.4	5.2	4.6	21.6	35.7	(*)156.9
	Int(*10000)	1.1	0.3	1.3	2.4	10.2	2.0	15.7	5.1	5.4	50.1	110.4	224.5
	Real	0.2	0.3	1.8	2.6	21.1	2.4	18.8	8.8	7.0	58.7	116.8	367.7

We can draw several rather straightforward conclusions from the measurements in Table 1. First of all, it is better to use boolean variables rather than

integer variables to represent the mapping itself. For almost all cases the mapping with boolean mapping variables ran faster than with integers.

Secondly, the mapping runs faster if the numerical values are modeled as integers rather than reals, and the runtime depends on the precision of the integer representation. For example, if real values are multiplied by 100 and rounded, the mapping runs quicker, however due to rounding errors the produced mapping might be incorrect (violating initial constraints) or not optimal. As shown in Table 1, this was observed in several cases for multiplication by 100 and once for multiplication by 1000.

Thirdly, the level of load on the network created by the task graph does not seem to be a critical factor influencing the mapping time. For the load balancing objective, among the instances with 15 tasks, the hardest one seems to be the instance with high load, whereas for instances of size 10, the low-load instance took the most time to solve. Moreover, for the traffic minimization objective, the mid-load instance appears to be the hardest among the instances of size 15.

The mapping for the traffic minimization objective is significantly faster than for the load balancing objective, on average about 30 times faster. This is probably caused by the way the load balancing objective was specified. The nested max() function results in many if-then-else statements, whereas the traffic objective is a simple sum.

5.2 Task-Node Mapping for the ATON Application

We evaluated our task-node mapping approach with the existing application 'Autonomous Terrain-based Optical Navigation' (ATON) [7], which is expected to run on the ScOSA platform in the future. ATON is a navigation system, currently developed at DLR. It is designed to provide a navigation solution for a precise landing on the Moon or other celestial bodies. It consists of the following sensors: an inertial measurement unit (IMU), two cameras, a laser altimeter and an (emulated) star tracker. The system was recently tested on an unmanned aerial vehicle (UAV) in a closed-loop scenario. The ATON software consists of several modules for absolute (crater navigation) and relative (feature tracking) navigation. The navigation filter combines the results of the sensors and software modules and provides a position, velocity and attitude estimation. The task graph for this application is shown in Fig. 4. We estimated the periods for non-sensor tasks using the approach described in Subsection 3.1.

The majority of the data is sent from Camera tasks to Undistortion tasks (camera images with high frequency), and from Undistortion to Crater navigation (undistorted images with lower frequency). The Crater navigation tasks receive more inputs than they are able to process. In practice, this means that when a Crater navigation task finishes, it starts processing the latest received image immediately, thus creating a full load on a node. To model this, we trigger the Crater navigation tasks on every fourth incoming message because the execution time of Crater navigation is slightly less than the time of arrival of four incoming messages.

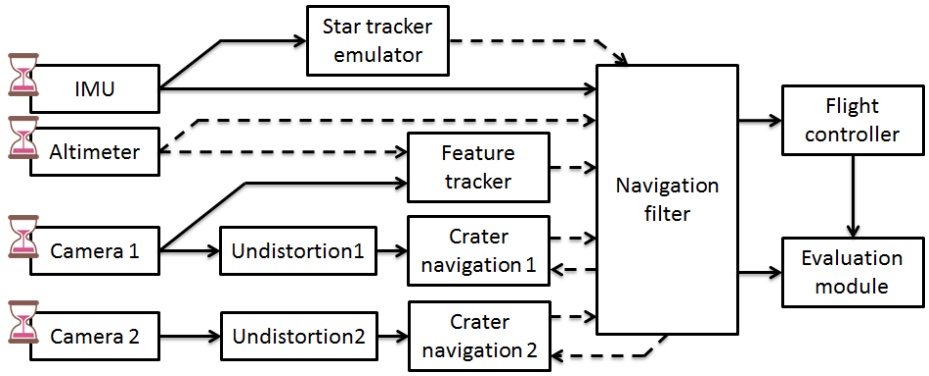

Fig. 4. Autonomous Terrain-based Optical Navigation (ATON) application task graph. Dashed lines show optional messages that are not required to start a task.

The mapping for the ATON task graph and our four-node example network from Fig. 1 is shown in Fig. 5. We calculated mappings for the two objectives discussed above: traffic minimization and load balancing. They took 0.4 and 1.8 seconds respectively. The load balancing is not an appropriate objective in this case because the Crater navigation tasks use all the capacity of their nodes, so the maximal node load, which is minimized, will always be the same. The traffic minimization, on the other hand, shows good results: the tasks with the most communication are colocated (Camera and Undistortion tasks).

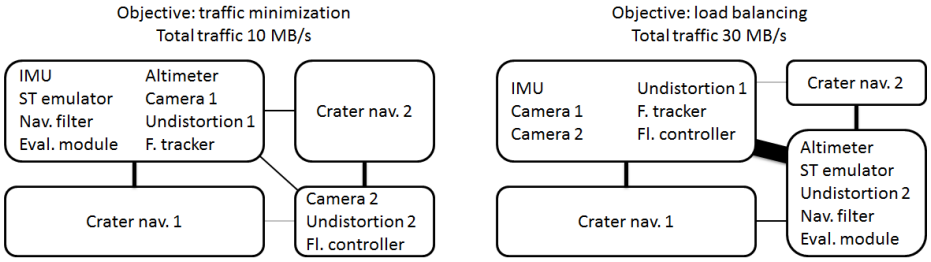

Fig. 5. ATON mapping to a four-node network with two different objectives. The width of lines between the nodes corresponds to the amount of exchanged data. The gray lines show that there is no data transfer between the nodes.

5.3 Remarks on Scalability

In order to evaluate the scalability of our approach, we generated a number of random task graphs of different sizes. We generated 10 task graphs for each problem size (5, 10, 15, 20 tasks) and took the average mapping time, excluding

few instances that were unsatisfiable (in which case the solving ran under a second). The graph of average mapping times is shown in Fig. 6. There is a significant variation of solving time even for the same problem size. For example, the time for load balancing with 15 tasks ranges from 1.4 seconds to 8.7 minutes.

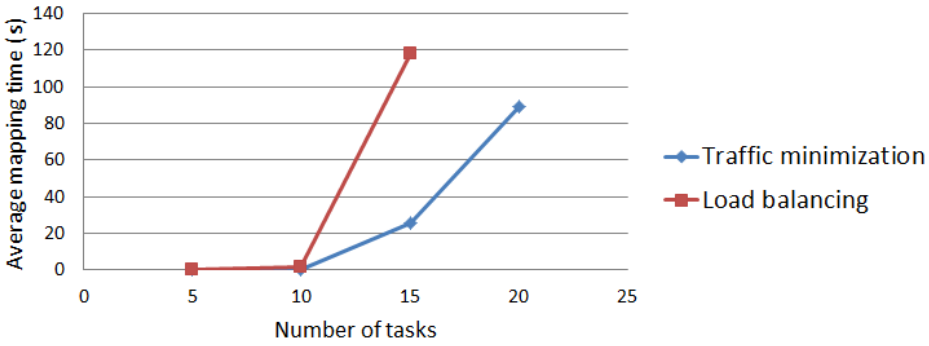

Fig. 6. Mapping time for different number of tasks with different objectives.

It is clear that the scalability of this approach is limited because the search space and the runtime grow exponentially with the number of tasks. For t tasks and n nodes there are n^t possible mappings, so even for rather simple cases with 4 nodes and 20 tasks we have to search within a large state space. Considering that some instances are mapped significantly faster than others, our best guess is that this approach can be practical for some instances of up to 30-50 tasks.

This is of course true for any exact optimization solution. When the problem instance is too large for an exact method, the only solution is to use approximate approaches such as search heuristics or genetic algorithms.

6 Conclusions and Future Work

We have demonstrated an application of SMT solving to produce a task-node mapping between tasks that run periodically in a distributed system, and the available processing nodes that are connected to a network with an arbitrary topology.

We formally defined the task-node mapping problem originating from our project ScOSA, showed how to translate this problem into an SMT problem, and solved it with the state-of-the-art SMT solver Z3.

In order to evaluate our approach, we used it to map the tasks of an existing application in a reasonable time. Additionally, we evaluated the scalability of the approach by running it on generated problem instances of different sizes.

As a result, the SMT-based approach proved to be suitable for the topology-aware mapping problem and practical for many problem sizes, which we suppose to have in ScOSA.

Our practical recommendations on SMT modeling to achieve some speedup are as follows. First, it appears to be better to use boolean encoding of an integer index variable rather than straightforward integer encoding. Second, it is advantageous to round numerical values to integers (however, with lower precision there is a risk of getting incorrect or suboptimal mapping).

A number of questions, however, remain for the future work. First of all, we need to compare our approach to the other approaches listed in Sect. 2. Secondly, we need to evaluate how accurately our model with periodic tasks reflects the actual system containing both time-triggered and event-triggered tasks. Additionally, it would be interesting to investigate if the solving time depends on the topology of the task and network graphs.

In the future, the task-node mapping will be extended with model checking to support the systems engineering for future ScOSA systems. For the mapping we consider a rather simple model of the system and the network. In the subsequent model checking, we are planning to analyze more complicated properties on a more precise system model such as message delivery times considering the routers' behavior and the routing protocol, time bounds on failure detection, absence of 'reconfiguration races' where different reconfigurations are triggered simultaneously, time bounds on the reconfigurations, application-specific properties.

Our long-term goal is to create a tool where the user inputs the network structure, the task graphs, requirements and constraints, and gets a collection of optimal mappings that satisfy the requirements, if such mappings exist.

References

1. Baruah, S.: Task partitioning upon heterogeneous multiprocessor platforms. In: Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004. pp. 536–543 (2004)
2. Biewer, A., Andres, B., Gladigau, J., Schaub, T., Haubelt, C.: A Symbolic System Synthesis Approach for Hard Real-Time Systems Based on Coordinated SMT-Solving. In: 2015 Design, Automation Test in Europe Conference Exhibition (DATE). pp. 357–362 (2015)
3. Bjørner, N., Phan, A.D., Fleckenstein, L.: νZ - An Optimizing SMT Solver. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol. 9035, pp. 194–199. Springer (2015)
4. Cheng, Z., Zhang, H., Tan, Y., Lim, Y.: SMT-based Scheduling for Multiprocessor Real-Time Systems. In: 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS). pp. 1–7 (2016)
5. Cruz, E.H.M., Diener, M., Pilla, L.L., Navaux, P.O.A.: An Efficient Algorithm for Communication-Based Task Mapping. In: 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. pp. 207–214 (2015)
6. Deveci, M., Rajamanickam, S., Leung, V.J., Pedretti, K., Olivier, S.L., Bunde, D.P., Çatalyürek, U.V., Devine, K.: Exploiting Geometric Partitioning in Task Mapping for Parallel Computers. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium. pp. 27–36 (2014)

7. Franz, T., Lüdtke, D., Maibaum, O., Gerndt, A.: Model-based software engineering for an optical navigation system for spacecraft. In: Deutscher Luft-und Raumfahrtkongress. Braunschweig, Germany (Sep 2016)
8. Glantz, R., Meyerhenke, H., Noe, A.: Algorithms for Mapping Parallel Processes Onto Grid and Torus Architectures. In: Proceedings of the 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. pp. 236–243. PDP '15, IEEE Computer Society, Washington, DC, USA (2015)
9. Hoefler, T., Jeannot, E., Mercier, G.: An Overview of Process Mapping Techniques and Algorithms in High-Performance Computing. In: Jeannot, E., Zilinskas, J. (eds.) High Performance Computing on Complex Environments, pp. 75–94. Wiley (2014)
10. Hoefler, T., Snir, M.: Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In: Proceedings of the International Conference on Supercomputing. pp. 75–84. ICS '11, ACM, New York, NY, USA (2011)
11. Lee, S.Y., Aggarwal, J.K.: A Mapping Strategy for Parallel Processing. IEEE Transactions on Computers C-36(4), 433–442 (1987)
12. Lei, T., Kumar, S.: A Two-step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture. In: Euromicro Symposium on Digital System Design, 2003. Proceedings. pp. 180–187 (2003)
13. Liu, W., Gu, Z., Xu, J., Wu, X., Ye, Y.: Satisfiability Modulo Graph Theory for Task Mapping and Scheduling on Multiprocessor Systems. IEEE Transactions on Parallel and Distributed Systems 22(8), 1382–1389 (2011)
14. Liu, W., Yuan, M., He, X., Gu, Z., Liu, X.: Efficient SAT-Based Mapping and Scheduling of Homogeneous Synchronous Dataflow Graphs for Throughput Optimization. In: 2008 Real-Time Systems Symposium. pp. 492–504 (2008)
15. Lüdtke, D., Westerdorff, K., Stohlmann, K., Börner, A., Maibaum, O., Peng, T., Weps, B., Fey, G., Gerndt, A.: OBC-NG: Towards a Reconfigurable On-board Computing Architecture for Spacecraft. In: Aerospace Conference, 2014 IEEE. pp. 1–13 (2014)
16. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08). Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008)
17. Potts, C.: Analysis of a linear programming heuristic for scheduling unrelated parallel machines. Discrete Applied Mathematics 10(2), 155 – 164 (1985)
18. Satish, N., Ravindran, K., Keutzer, K.: A Decomposition-based Constraint Optimization Approach for Statically Scheduling Task Graphs with Communication Delays to Multiprocessors. In: 2007 Design, Automation Test in Europe Conference Exhibition. pp. 1–6 (2007)
19. Tendulkar, P., Poplavko, P., Maler, O.: Symmetry Breaking for Multi-criteria Mapping and Scheduling on Multicores, pp. 228–242. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
20. Wang, L., Li, Z., Song, M., Ren, S.: A Genetic Algorithm based Approach to Maximizing Real-Time System Value under Resource Constraints. In: 2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC). pp. 285–294 (2012)
21. Yang, L., Liu, W., Jiang, W., Li, M., Yi, J., Sha, E.H.M.: Application Mapping and Scheduling for Network-on-Chip-Based Multiprocessor System-on-Chip With Fine-Grain Communication Optimization. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 24(10), 3027–3040 (2016)